Red Green Refactor

Aleksander Zubala

@alekzubala
github.com/azubala

Question?

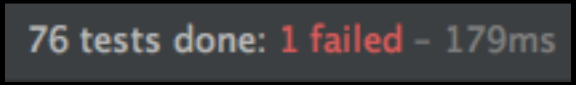
- Did you experience while TDDing:
  - test didn’t work as you expected
  - test passed but app functionality was broken
  - test failed but app functionality was correct
  - assertion was always true/false no matter the circumstances

Red Green Refactor

Red Green Refactor

- Development in short, repeatable cycles
- Technique based on 3 steps to verify test that you're writing
    - RED: write test that fails
    - GREEN: make it pass
    - REFACTOR: improve your implementation
- Constantly forming hypotheses and checking them
- As you progress changes are covered

RED



```
76 tests done: 1 failed - 179ms
```

- Think for a while what piece of code move your project towards completion
- Write a short test
  - tested object might not exist
  - method might not be implemented
  - AppCode helps to quickly implement/create missing classes or methods
- Execute tests, check if the **test is failing**

GREEN

All Tests Passed

- Write a production code in your project:
  - the previous test must pass
  - do not focus on the quality of the code
  - you can even hardcode to achieve the passing test
- Execute test, check if the **<u>test is passing</u>**
- Now you have a proof that the test is testing the right thing

REFACTOR

- Take a deep breath, all of your tests are passing:)
- Go back to the code you've just written, see what can be improved
- Don't be afraid to change the code, tests will quickly catch mistakes
- Focus on the code duplication (DRY)
- Easy to spot architectures flaws
- No idea how to improve - iterate through couple of RDGs
- REPEAT!

Photo Stream

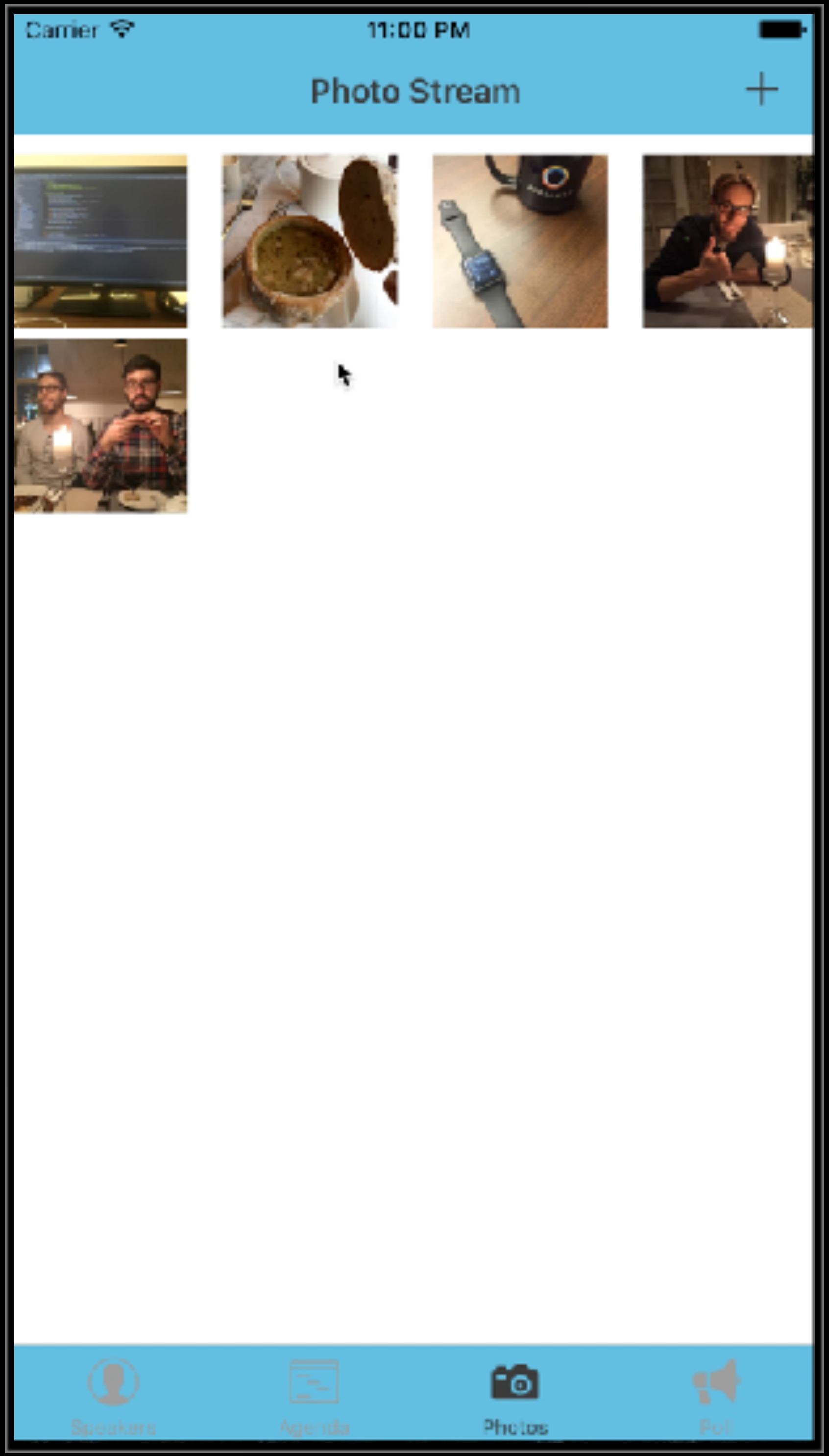


Photo Stream

- UI
  - `PhotoStreamViewController`
  - `StreamItemViewController`
- Model
  - `StreamItem`
  - `StreamItemCreator`
  - `StreamItemUploader`
  - `StreamItemDownloader`

What's up?

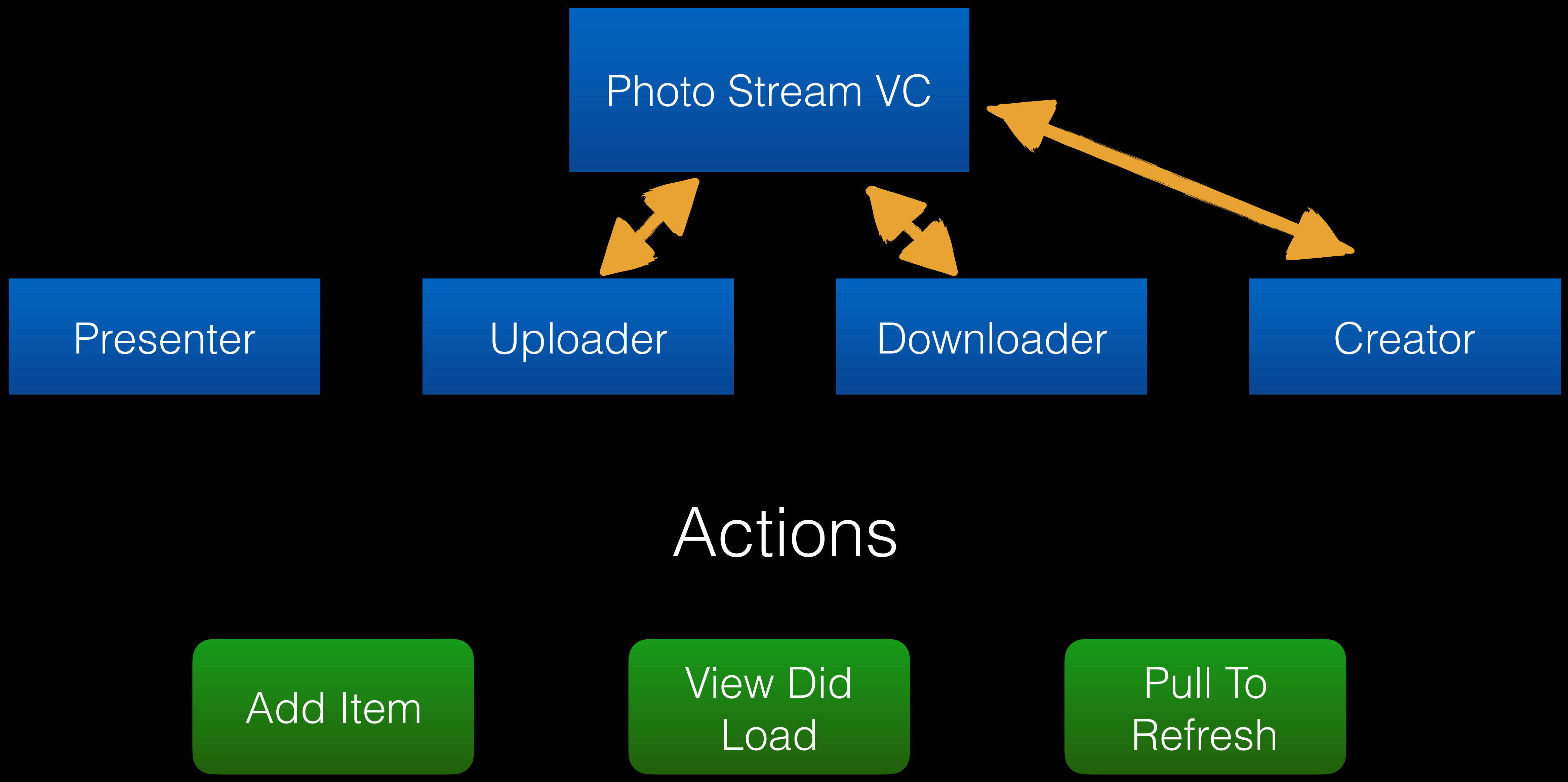


Photo Stream VC
Presenter
Uploader
Downloader
Creator
Actions
Add Item
View Did Load
Pull To Refresh

Let’s code!

Tasks for today

Task 1: Reload collection view after item was created

Task 2: UX bug when creating item

Task 3: Item title

Task 1: Reload after creation

- **What?**
  - When items was created it does not appear
- **How?**
  - `PhotoStreamViewController -> ItemCreatingDelegate`

```
func creator(_ creator: ItemCreating, didCreateItem item: StreamItem)
```

  - Insert newly created item to `streamItems`
  - Reload `UICollectionView` (use `UICollectionViewFake` to test)
- **Verify**
  - Checkout branch to compare: `photo-stream-task-1`

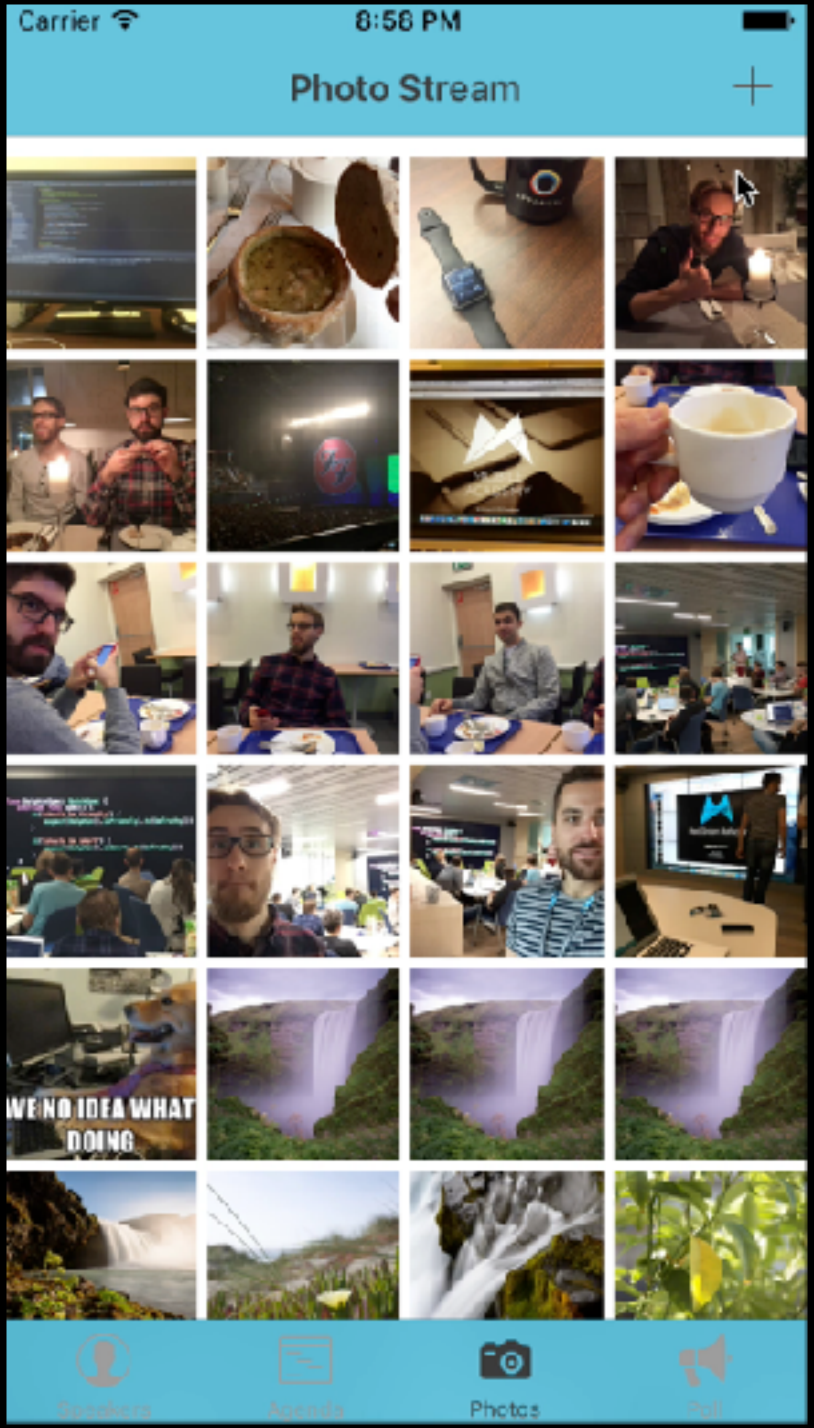
Task 2: UX bug when creating item

- **What?**
  - When user presses add item button and only Photo Library or only Camera is available, action sheet with single option is presented
- **How?**
  - Checkout class which provides available source types
  - When single source available do not present action sheet
  - When no source types available inform delegate about error
- **Verify**
  - Checkout branch to compare: `photo-stream-task-2`

Task 3: Item title

- **What?**
  - All stream items have the same, hardcoded title
  - Implement UI which allows user to provide title of the `StreamItem` when created
- **How?**
  - Introduce additional step in `StreamCreator` to provide title
  - You can use `UIAlertController` with text field to prompt user to provide title
  - Modify code which creates `StreamItem` to use provided title
- **Verify**
  - Checkout branch to compare: `photo-stream-task-3`

Task 3: Item title



@alekzubala #TDDCraftConf

Gist with instructions: `goo.gl/PvST7m`

Behaviour to implement:

- dismiss image picker (use `ViewControllerPresenterFake`)
- presented alert controller (use `ViewControllerPresenterFake`)
- alert controller should have:
  - title: "Provide item title:"
  - one action: "OK" (use `AlertActionFactoryFake`)
  - one text field (check `textFields` property on `UIAlertController` ; use `addTextFieldWithConfigurationHandler:`)
- when action is executed
  - picked image should be scaled (use fake `ImageManipulatorFake`)
  - picked image should be converted to `Data` (use fake `ImageManipulatorFake`)
  - inform delegate about item creation (use `TestStreamItemCreatorDelegate`)
  - stream item created with scaled image (check captured item in fake delegate)
  - stream item created with title from text field (check captured item in fake delegate)

Gist with instructions: `goo.gl/PvST7m`

Tips & Tricks

- Don’t spend too much time on red/green cycle
- Try not to achieve fully functional feature in single cycle
- Always take a second to think about refactor
- Refactor also your specs (be careful though)
- When facing code which seems not testable - break dependencies, extract functionalities
- Swift: use protocols - easier to fake behaviours

Recap

- First test must fail, then pass, then you refactor
- Baby steps - a lot of small cycles
- Spend most time in Refactor stage
- Execute tests often: when something goes wrong, it's easy to identify what was the cause
- Makes you write testable code - you have to think about design ahead
- Once feature is done, it's covered with tests

Thanks!